NPS-CS-03-004
February 2003


The Center for INFOSEC Studies and Research

white paper

# Policy Enforced Remote Login

Thuy D. Nguyen and Timothy E. Levin

Center for Information Systems Security Studies and Research
Computer Science Department
Naval Postgraduate School
Monterey, California 93943

# Policy Enforced Remote Login

Thuy D. Nguyen and Timothy E. Levin

Center for Information Systems Security Studies and Research
Computer Science Department
Naval Postgraduate School
Monterey, California 93943

Abstract:

*This document describes enhancements made to the popular OpenSSH authentication service to restrict the execution of OpenSSH processes by applying a ring-based program execution policy. We also apply a label-based mandatory access control (MAC) policy to limit a user's login shell to run at a specific security level within the user's authorized security clearance range. While still rudimentary, these enhancements illustrate the usefulness of a ring-based execution mechanism for restricting program behavior.*

## Introduction

This document describes the enhancements made to the popular OpenSSH implementation to provide a policy-enforced remote user authentication service. OpenSSH is a client-server application that provides a range of remote authentication and other security services. On the server side, OpenSSH uses the privilege separation approach to prevent privilege escalation by containing privileged and unprivileged operations in separate processes. The Policy-Enforced Remote LogIn (PERLI) service extends the OpenSSH privilege separation mechanism by applying a ring-based program execution control policy to further restrict the execution domains of the OpenSSH processes. PERLI also applies a label-based mandatory access control (MAC) policy to limit a user's login shell to run at a specific security level within the user's authorized security clearance range.

The PERLI extension was added to OpenSSH 3.5. It runs on a modified version of OpenBSD 3.1 that supports MAC enforcement and Ring protection described in [5], [2] and [3]. The OpenSSH implementation of privilege separation is described in [1]. PERLI was developed specifically to illustrate domain separation capabilities in support of the Homeland Security Research and Technology program of the Department of Justice.

## Background

The Secure Shell (SSH) protocol suite was designed to provide secure remote login and other secure network services over an unprotected network. OpenSSH is a free, open source implementation of the SSH protocol suite. The SSH client program (*ssh)* allows a user to login and execute commands on a remote server over encrypted channels. The SSH daemon (*sshd*) authenticates the user and, if the authentication is successful, spawns a child process to execute the requested login shell or commands. The SSH daemon is usually started by the *init* program and runs with superuser privileges. Without privilege separation, the SSH daemon's child process will also run with the same set of special privileges which can be abused to compromise the integrity of the system. With privilege separation, the child process will only run with the privileges for which the current user is authorized.

Privilege separation is achieved by splitting a process into two parts: a privileged parent and an unprivileged child. The parent process monitors the progress of the child process and performs privileged operations for the child process. The child process performs networking functions and user requests. Figure 1, borrowed from [1], illustrates the process architecture of the privilege-separated OpenSSH.

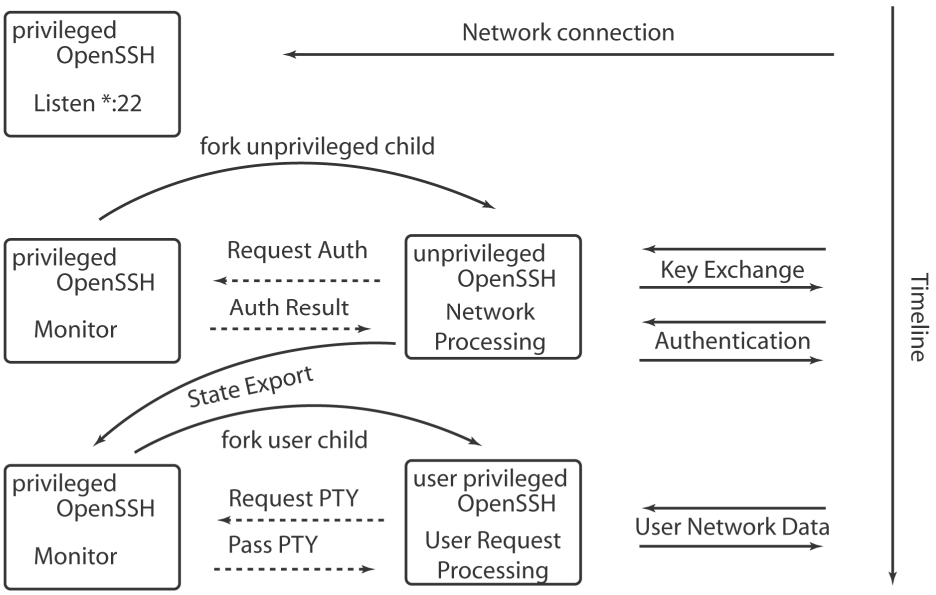


Figure 1

After a TCP connection is established between the client and the server, the SSH Listener forks a privilege Monitor process which, in turns, forks an unprivileged child process to handle user authentication requests. Because the user is not yet authenticated, the unprivileged child process at this stage runs as a special "sshd" user with no login and file creation capability. The pre-authentication child process is ephemeral. The Monitor

process retains superuser privileges throughout its lifetime in order to perform privileged operations. After the user is authenticated, the Monitor process forks another unprivileged child process to execute the requested functions. The post-authentication child process runs with the privileges assigned to the authenticated user.

The privilege-separated OpenSSH architecture is a natural candidate to demonstrate that MAC enforcement and Ring protection can make application-level security mechanisms more secure. On operating systems that support MAC and/or Ring policies, the SSH daemon needs to spawn the post-authentication child process at the appropriate security level and execution domain that are within the user's security clearance. With these protections in place, even if the SSH modules or the program that they invoke became corrupted, the damage they can cause will be limited to the level and domain to which they are assigned.

The SSH protocol defines an authentication method called "keyboard-interactive". It is a relatively new authentication method for SSH and is defined in [4]. The IETF draft describes it as follows:

> "This method is suitable for interactive authentication methods which do not need any special software support on the client side. Instead all authentication data should be entered via the keyboard. The major goal of this method is to allow the SSH client to have little or no knowledge of the specifics of the underlying authentication mechanism(s) used by the SSH server."

This method is suitable for PERLI since PERLI requires the user to specify a password and a session level interactively. Furthermore, OpenSSH already uses this method to implement challenge-response and one-time password authentication mechanisms.

## Concept of Operations

PERLI provides MLS-aware remote login functionality. In a distributed security architecture such as [5], it can be used as an enhanced trusted path mechanism to ensure that the user is interacting directly with the security control mechanism of the operating system, and to subsequently provide authentication, session-level negotiation, and other security interactions. In this scenario, lesser-privileged users who have no access to high assurance hardware authentication devices can use the COTS SSH client software on their desktops, laptops, even PDA's to remotely log onto the PERLI-enabled server to access information that is available at the authenticated session level.

PERLI can also be used to demonstrate the concept of emergency response domain separation. In an emergency network mode, network activities of less privileged domains are temporarily halted. After the emergency situation is resolved, less-critical functions are gradually enabled to bring the system back to a normal state. Logic in the security management function will determine whether PERLI is to be enabled or disabled while a system is in this transitional mode. If enabled, PERLI will run in a designated execution domain and only users who are authorized to execute programs in that domain can use PERLI to log onto the system. Once authenticated by PERLI, the user can use the encrypted channels provided by the SSH transport protocol to perform the network functions required to manage and restore the system.

## Implementation

The current PERLI implementation only affects the SSH server code. No code changes are required on the client side. Code modifications were made in two stages. The initial development was done on a modified version of OpenBSD 3.1 in which root is exempt from all MAC checks. This version does not support the Ring policy as described in [3]. The reason for using this version is to gain familiarity with the OpenSSH source code and usage.

The second development phase was done on a separate modified version of OpenBSD 3.1 in which the Ring policy is introduced and all users (including root) are subject to MAC and Ring enforcements. This prototype system supports four execution rings, ranging from 0 to 3. The association between rings and execution domains is as follows:

| | |
|---|---|
| Unprivileged Application Domain | Ring 3 |
| Privileged Application Domain | Ring 2 |
| Admin Domain | Ring 1 |
| OS Domain | Ring 0 |

Similar to the privilege separation approach, application programs on a ring-enforced system are separated into two execution domains to provide finer control of program integrity and protection of critical application data. Programs assigned to a lesser privileged ring (e.g., Ring 3) will be unable to execute or access objects allocated in a more privileged ring (e.g. Ring 2). Within this paradigm, the ring allocation of the PERLI processes is as follows:

| | |
|---|---|
| Post-authentication Child | Ring 3 |
| Pre-authentication Child | Ring 3 |
| Monitor | Ring 2 |
| Listener | Ring 2 |

The following modifications were made to the server code.

1. New module to handle PERLI-specific authentication functions. These include invoking the underlying keyboard-interactive handler to prompt the user for the password and desired session level, authenticating the user password, and verifying that the specified session level is within the user's clearance obtained from the system's clearance database. By default, PERLI only allows remote users (including root) to login and run a single level session. This restriction can be lifted for the root user by enabling a special server configuration option that allows root to login with a range. If no session level is given, the user's default session level is used.
2. Modification to pre-authentication processing. Additional logic was added to the Monitor process to change its execution domain to Ring 2, and to set the security level and execution domain of the pre-authentication child process to the default session level of the special "sshd" user and Ring 3, respectively.
3. Modification to post-authentication processing. The Monitor process was also changed to set the security level and execution domain of the post-authentication child process to the requested session level and Ring 3, respectively.

In the original OpenSSH implementation, privilege separation ends after the pre-authentication phase if the remote user is root. With respect to user privileges, it is redundant for the Monitor process to fork a post-authentication child process that will end up having the same privileges as the Monitor itself. But with respect to security levels, a system may be required to restrict root to run with a lesser security level if root logs in remotely. Hence, the original logic was modified so that the post-authentication child process will be created for all users.

PERLI is configured at both compile time and runtime. At compile time, the "PERLI" compiler toggle must be set and the default "BSD_AUTH" toggle must be unset. The required runtime configuration options are described below. Furthermore, PERLI has only been tested with SSH protocol version 2.

## User Interface and Configuration

The following options must be set in the various configuration files in order to use PERLI. Options with asterisk (*) are newly added for PERLI. Items with double asterisks (**) are existing options with newly added values.

1. SSH client configuration file (/etc/ssh/ssh_config)
   - PreferredAuthentications=keyboard-interactive
2. SSH server configuration file (/etc/ssh/sshd_config)
   - UsePrivilegeSeparation=yes
   - PERLIAuthentication=yes*
   - PermitRootLogin=perli-with-range**

   This option should only be used if root is allowed to login remotely with a range.
3. Clearance database (/security/bibClearance, /security/blpClearance, /security/rinClearance)

These files must be configured to specify the security clearance of the users who are authorized to login remotely as well as the security clearance of the special "sshd" user. The "sshd" user is added to the system's user database as part of the OpenSSH installation procedure.

## Testing With SSH Clients

PERLI was tested with the following open source client programs:

1. PuTTY, a Windows SSH client program, running on a Windows 2000 laptop;
2. OpenSSH client program running on a Red Hat Linux desktop;
3. OpenSSH client program running on a Compaq iPAQ 3600 handheld computer.

Figure 2 is a screenshot of a PERLI session using PuTTY.

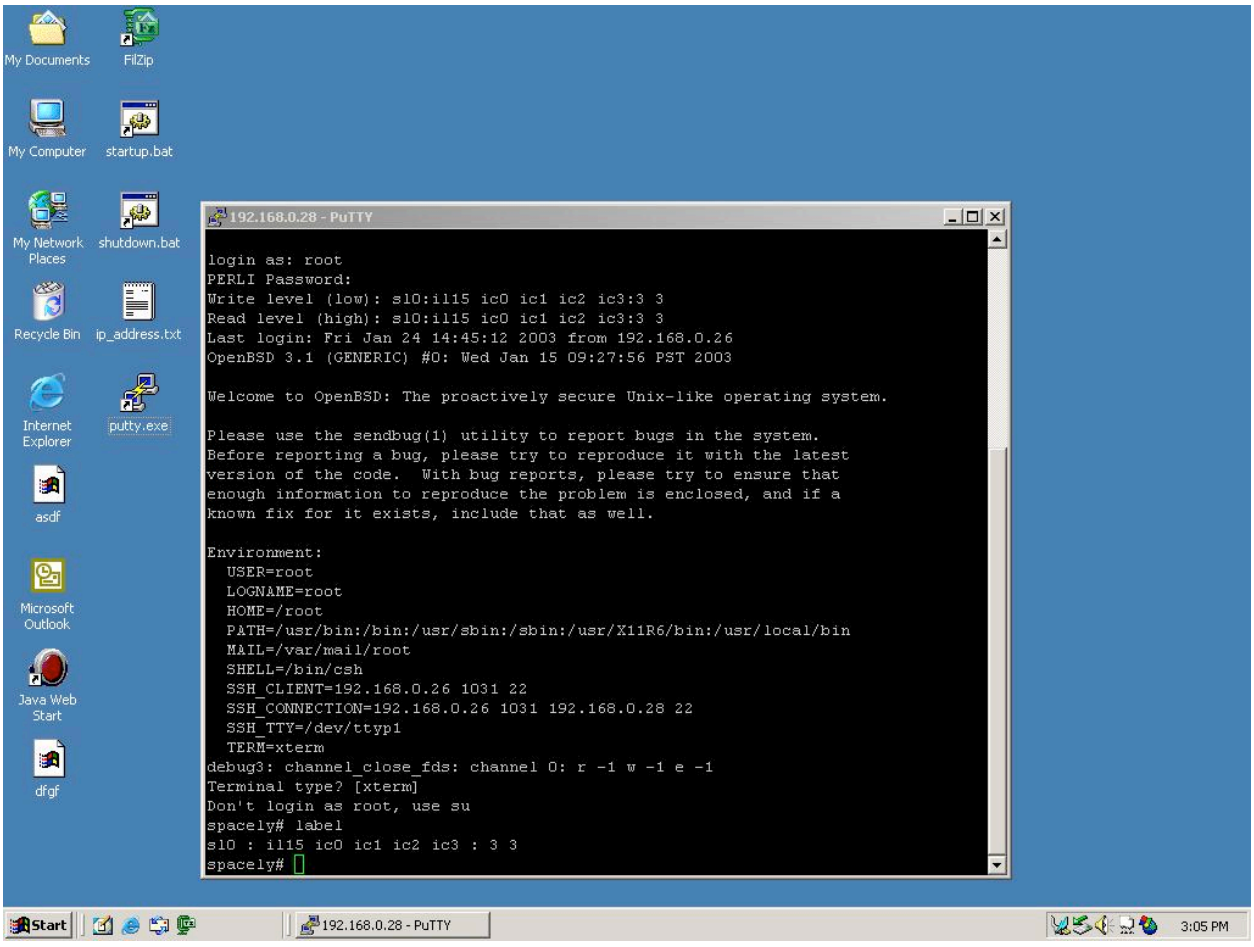


Figure 2

## Summary

The current PERLI implementation, while rudimentary, illustrates the usefulness of a software-based ring execution policy for restricting program execution. Lots of work is still required to make PERLI more robust and compatible with existing authentication methods that OpenSSH supports. Future work includes audit enhancements, applying MAC and Ring enforcements to the port tunneling capabilities that OpenSSH also provides, and investigating how OpenSSH with PERLI extension can be used in a distributed MLS architecture, such as [5].

## References

[1] Niels Provos. Preventing Privilege Escalation, CITI Technical Report 02-2, August 5, 2002.

[2] Paul C. Clark. Policy-Enhanced Linux, 23rd National Information Systems Security Conference, October 16-19, 2000.

[3] Paul C. Clark, et al, Execution Policies Research and Implementation, NPS Technical Report NPS-CS-03-003, February, 2003.

[4] Frank Cusack, Martin Forssen. Generic Message Exchange Authentication For SSH, Internet Draft, <draft-ietf-secsh-auth-kbdinteract-04.txt>, October 2, 2002.

[5] Cynthia E. Irvine, David J. Shifflett, Paul C. Clark, Timothy E. Levin, George W. Dinolt, MYSEA Security Architecture, NPS Technical Report NPS-CS-02-006, May 2002